Software Engineering Institute

SOFTWARE ACQUISITION PLANNING GUIDELINES

CMU/SEI-2005-HB-006

Editor:
William E. Novak

Contributors:
Julie B. Cohen
Anthony J. Lattanze
Linda Levine, PhD
William E. Novak
Patrick R. H. Place
Ray C. Williams
Carol Woody, PhD

December 2005

Acquisition Support Program

20060515171

Carnegie Mellon

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Table of Contents

Acknowledgments

Many people have contributed to creating these guidelines, both directly and indirectly. It would not have been possible to provide guidance on such a wide range of software acquisition topics without the combined expertise and prior work of many others. We would like to thank our sponsor, the United States Army Strategic Software Improvement Program (ASSIP), for the opportunity to perform this work.

We would like to thank the Software Engineering Institute (SEI) research programs that have researched, developed, and published the work and ideas represented here, including the Acquisition Support Program, Dynamic Systems Program, Networked Systems Survivability Program, Product Line Systems Program, and the Software Engineering Process Management Program.

In particular, the editor would like to thank Joseph Elm, Julie Cohen, Jeannine Siviy, Wolfhart Goethert, Mary Catherine Ward, Ray Williams, and the many other authors and contributors to the SEI's *Software Acquisition Survival Skills* course, on which several of the guidelines are based. We are also indebted to Lisa Brownsword, Patricia Oberndorf, and Dr. Carol A. Sledge for their work in developing the SEI's *COTS-Based Systems for Program Managers* course.

We would also like to thank all of the people within the SEI who provided technical review and comment, including: Cecilia Albert, John Bergey, Julie Cohen, Robert Ferguson, Donald Firesmith, Mary Ann Lapham, B. Craig Meyers, Patricia Oberndorf, Patrick Place, John Robert, and Ray Williams. Due to their efforts and expertise, this is a much better document.

Finally, the authors are most grateful for the essential help we have received from our editor, Susan Kushner, and our document designers, Stacy Mitchell and Melissa Neely. Their efforts have resulted in a clearer and simpler document.

Abstract

Guidance about acquisition planning and strategy is scattered by topic throughout many different books, reports, presentations, and Web sites. This handbook presents guidance for acquisition planning and strategy topics in a condensed form, and references the primary resources available for each topic.

The topics for the guidelines in this handbook represent selected areas in which the SEI has conducted significant research. The guidelines are organized by the acquisition strategy considerations categories from the Defense Acquisition University's *Defense Acquisition Guidebook* that describe the principal areas of consideration for planning an acquisition strategy. Guidance is offered on 13 topics, including open systems approach, commercial off-the-shelf (COTS)-based systems, software architecture, software sustainment, requirements development, requirements management, operational information assurance, information assurance for COTS sustainment, information asset protection, software testing, software risk management, software metrics, and software-based award fees.

This handbook is intended to provide program managers and project management office staffs with recommendations and resources for addressing different aspects of their acquisition strategy. It illustrates acquisition challenges with program scenarios, suggests actions that should be taken, and identifies danger signs to look for when evaluating progress. A basic knowledge of both software development and acquisition practices is assumed.

Introduction

"Good judgment is usually the result of experience. And experience is frequently the result of bad judgment. But to learn from the experience of others requires those who have the experience to share the knowledge with those who follow."

—Barry LePatner

BACKGROUND AND RATIONALE

The software acquisition practitioners assigned to government Program Management Offices (PMOs) have policy and regulations they must follow, but they also need concise, practical guidance to assist them. Good information on acquisition planning and strategy is available, but it is scattered by topic across many different publications. Program managers (PMs) and their staffs want to know the practical "lessons learned" of acquisition planning with enough context to determine their applicability to their program, and sufficient detail to be actionable.

OBJECTIVES

The key objectives of the acquisition planning guidelines presented in this handbook are to

- create a collection of acquisition planning knowledge that will be a useful resource to busy acquisition practitioners
- condense existing acquisition planning guidance from the SEI into a concise format that is easily accessible
- focus on the issues of software aspects of acquisition planning (rather than on the entirety of acquisition)
- offer specific and actionable guidance

APPROACH

The guidance contained in this handbook has been collected from SEI publications and courses, and edited into condensed guidelines. It reflects a broad scope of experience, and provides lessons and best practices gained from the SEI's work with acquisition programs throughout the United States government.

A key element in presenting practical guidance is to ground it in a real-world situation. In this handbook, each acquisition guideline is associated with a short scenario that offers a compelling rationale for the guidance.

In an effort to create a relevant and accessible document, each guideline is presented in a format that focuses on recognizing and preventing problems. Each guideline is supported by

1) a guideline statement
2) a scenario that exemplifies the practice or issue
3) an analysis of the scenario and discussion of the broader issues
4) a set of "Required Actions" that recommend preventative or corrective actions to be taken
5) "Danger Signs" that a PMO should look for as symptoms that a program is going "off course"
6) links and references to practical "how to" information

TARGET AUDIENCE

The intended audience for this handbook is the PMO staffs that acquire software-intensive systems. The handbook is meant to provide program managers and PMO staffs with ideas and resources for addressing an aspect of their acquisition strategy, specifying actions that should be taken, and identifying danger signs to look for when evaluating progress. A basic knowledge of both software development and acquisition practices is assumed.

ACQUISITION PLANNING AND STRATEGY

According to the Defense Acquisition University (DAU), an *acquisition strategy* is a high-level "road map for program execution from program initiation through post-production support."[1] A strategy is iterative, it varies in level of detail for different phases of the acquisition, and it should be updated periodically as the program's circumstances change. DoD Instruction (DoDI) 5000.2 requires that an acquisition strategy be delivered for all programs at Milestones B and C and at the Full-Rate Production Deployment Review.[2]

In contrast, an *acquisition plan* is a Federal Acquisition Regulation (FAR) document that is typically required for one contract within an acquisition. The *DAU Glossary* claims that it reflects "...the specific actions necessary to execute the approach established in the approved acquisition strategy and guiding contractual implementation."[3] However, the creation of an acquisition plan should not be confused with the activity of *acquisition planning*, which includes the creation and maintenance of an acquisition strategy.

Definitions of these terms are provided by the DAU *Defense Acquisition Guidebook*, but they are not as specific as we might wish:

- *acquisition planning*: The process by which the efforts of all personnel responsible for an acquisition are coordinated and integrated through a comprehensive plan for fulfilling the agency need in a timely manner and at a reasonable cost. It is performed throughout the life cycle and includes developing an overall acquisition strategy for managing the acquisition and a written Acquisition Plan.[4]

1 Defense Acquisition University. *DAU Acquisition Support Center Acquisition Strategy Page. https://acc.dau.mil/simplify/ev.php?ID=24229_201&ID2=DO_TOPIC*

2 Department of Defense. *DoD Instruction Operation of the Defense Acquisition System (DoDI 5000.2).*

3 Defense Acquisition University. *DAU Glossary, Version 11, plus Updates.*

4 Defense Acquisition University. *Defense Acquisition Guidebook.*

- *acquisition strategy*: A business and technical management approach designed to achieve program objectives within the resource constraints imposed. It is the framework for planning, directing, contracting for, and managing a program. It provides a master schedule for research, development, test, production, fielding, modification, postproduction management, and other activities essential for program success. The acquisition strategy is the basis for formulating functional plans and strategies (Test and Evaluation Master Plan (TEMP), Acquisition Plan (AP), competition, systems engineering, etc.).[3]

The following statement from the Air Force *Guidelines for Successful Acquisition and Management of Software-Intensive Systems*[5] provides perhaps the best description of an acquisition strategy:

"Acquisition strategy has been defined as a master plan, a road map, a blue print, and a plan-to-plan-by to achieve program goals and objectives. Your acquisition strategy serves as a means for reducing the odds of program defeat through the organized preparation of a plan to minimize software risk. It serves as a guide to direct and control the program, and as a framework to integrate those functional activities essential to fielding a totally operational system—not just pieces of hardware and software. The conceptual basis of the overall plan—the objective—is what you must follow during program execution. It also serves as the basis for all program management documents, such as the Acquisition Plan, the Test and Evaluation Master Plan (TEMP), the Integrated Logistics Support Plan, and the Systems Engineering Master Plan (SEMP)."

GUIDANCE SELECTION AND ORGANIZATION

The DAU *Defense Acquisition Guidebook* defines and describes in detail a set of 18 *acquisition strategy considerations,* which are the principal areas of consideration to be made when planning and developing an acquisition strategy. We have organized these guidelines into the *Defense Acquisition Guidebook* categories because they represent the way the DoD views acquisition strategy. The guidelines presented in this handbook address one or more of these acquisition strategy considerations. The list below shows the guidelines grouped according to the acquisition strategy consideration they support.

- **Modular Open Systems Approach**
  Open Systems Approach
  Commercial Off-The-Shelf (COTS)-Based Systems
  Software Architecture

- **Product Support Strategy**
  Software Sustainment

- **Capability Needs – Capability Areas**
  Requirements Development
  Requirements Management

5 Department of the Air Force, Software Technology Support Center. *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems, Version 3.0.*

- **Information Assurance**
  Operational Information Assurance
  Information Assurance for COTS Sustainment
  Information Asset Protection

- **Integrated Test and Evaluation**
  Software Testing

- **Risk Management**
  Software Risk Management

- **Systems Engineering – Metrics**
  Software Metrics

- **Systems Engineering – Performance Incentives**
  Software-Based Award Fees

The topics for the acquisition planning guidelines included in this handbook all represent areas in which a significant amount of work has been done at the SEI. They were selected based on the following criteria:

- the importance, or level of impact, of the guideline on acquisition success
- relevance to the area of software acquisition
- relevance to the PMO, rather than to the contractor
- the degree to which the information provides useful, actionable guidance
- the degree to which the information is broadly applicable across acquisition programs

This handbook is not complete or comprehensive with respect to acquisition planning guidance. Our intent is to provide an initial set of guidelines that offers a useful starting point for software-intensive acquisition programs. If this type of collected guidance is well received and proves to be useful, this handbook could be the basis for future work to expand this set of guidelines to provide more detailed guidance, or cover additional subject areas.

GUIDELINE 1: Open Systems Approach

Contributor: Patrick R. H. Place

1

GUIDELINE

Successfully using an open systems approach requires carefully selecting from the applicable standards, gaining expertise in conformance management, and adopting a willingness to invest more up-front to achieve long-term benefits.

SCENARIO

A program that was constructing a "system of systems" (multiple interoperating systems) issued a mandate that all constituent systems would be required to "use CORBA," the Common Object Request Broker Architecture. The program then learned that because the CORBA standard was loosely specified, there could be different implementations of CORBA, even though they all formally "conformed" to the standard. The CORBA standard—like most standards—had sections in it where choices were left to the developers implementing it. The result was that one contractor built one product with one set of choices and another contractor built another very different product using different choices, and yet in the end *both* conformed to the CORBA standard. Unfortunately, the different implementations could not work together.

ANALYSIS

The *Defense Acquisition Guidebook* (DAG) defines an open system as "...a system that employs modular design tenets, uses widely supported and consensus based standards for its key interfaces, and is subject to validation and verification tests to ensure the openness of its key interfaces."

The point of the open systems scenario above is that it is critical to know the *details* of the standards you are selecting and to look—*ahead of time*—for potential problems. This is not easy to do.

Standard interfaces are only the first part of an open systems approach. Equally important is an Open Systems Architecture for the system that provides a structure in which to *place* the standards. Then you need to *examine* those standards, specifying for *your* system how the standards fit together, what options of the standards will and will not be used, how implementation-specific parameters will be set, and so on. Some specific cautions regarding the use of open standards

- The use of standards can give a false sense of security. Simply adhering to the standard does not guarantee that all components will interface well, even if it can be said that they all conform to the standard.
- A standard may be inconsistent in its definition, resulting in conformance in theory, but non-conformance in reality. Similarly, a standard that is not fully specified can also create conformance issues.

- Who selects standards can have major implications, especially in a performance-based acquisition environment. For example, the PMO may tend toward selecting the most robust and comprehensive standard, while the contractor may be satisfied with a "least common denominator" standard with minimal conformance requirements. It is the system integrator who will be responsible for the overall operation of the system, and thus they should be responsible for standards selection, whether it is the government or a contractor.

Standards bodies can be biased since members pay to participate. More money means more control over the standard, which can advance a specific company's position. Be aware of such biases if a commercial implementation of the standard will be purchased.

The variety of competing standards makes choosing among them more difficult.
Whether the selection of standards is made by the contractor or the PMO, the PMO should ensure that the standards are compatible with one another, and that a system composed of components conforming to those standards is both achievable and desirable—in other words, it will perform acceptably.

- *Specifying*: Each standard that is selected affects the system architecture. Over-specifying the standards can lead to systems with sub-optimal designs. Under-specifying the standards can lead to systems that do not interoperate with other systems.
- *Type of Standard*: Try to choose standards that are widely supported formal standards, as they are more likely to be stable, and of longer term value than *de facto* standards that may change more frequently.
- *Maturity*: Try to select standards that are neither in the earliest stages (when they may still change substantially) nor at the end of their life (where they may soon be obsolete). You can often determine the age of a standard by evaluating the number of commercial products supporting it.
- *Maintenance*: As standards evolve, the program manager, along with the other stakeholders, must determine when it is appropriate to adopt a new (and possibly incompatible) version of the standard. This transition requires supporting multiple versions of the standard until all interacting systems can be upgraded, which introduces configuration management issues.
- *Product Standards*: Select standards that define interfaces between products and avoid standards that mandate the use of particular commercial products. This avoids changes in the "standard" due to upgrades of a product.

Develop expertise in conformance management.

- *Testing*: Using standards successfully requires either testing a component that is an implementation *of* the standard itself, or testing a component for conformance *to* the standard (where the application uses the standard to interface to another component or system). Testing implementations for conformance can be done by applying a test suite ensuring that each function of the interface behaves according to the standard. Testing applications for conformance is more difficult, and requires inspection of the source code. If the source code isn't available, exhaustive testing may be the only alternative.
- *Profiles*: Since systems often do not need all of the capabilities within a standard, a *profile* that identifies chosen subsets and options of a set of one or more standards may be useful. Using a new or existing profile can reduce the cost of conformance management and can increase the number of commercial products available for use by the system.

Expect changes in schedules and costs.

Increased up-front costs for longer term benefits: When using open standards, the initial cost increases, while the long-term costs may decrease. The initial increase is due to 1) performing the selection of standards and profile development, 2) completing a market survey to find conforming commercial items, 3) setting up conformance certification testing, and 4) training personnel to adopt an open systems strategy. Also, some architectural decisions depend on the use of standards, so these decisions must be made earlier. The long-term benefits can include 1) being able to *purchase* commercial items that meet the standards, 2) competition between vendors supplying conforming products and support, 3) some conformance testing that may have already been performed by the vendor. Also, contractors who are familiar with using open systems will not need training or time to become experienced with the approach.

REQUIRED ACTIONS

For choosing appropriate standards

- Where interoperability is the goal of the system, form working groups with the other interoperating systems to choose appropriate interface standards.
- Develop a clear understanding of the role the standards play in the overall system.
- Clearly state which standards are mandatory (such as, any that interface to external systems).
- When possible, recommend formal standards, but accept *de facto* standards when it is appropriate or necessary. This decision should be part of the business case analysis used in the standards selection process.
- Choose commercial-based rather than government-based standards whenever possible.
- Understand the contractor's justification for the selection of each standard.

For determining conformance

- Spend conformance effort on the parts of the standard that are relevant to your system.
- Use existing profiles if possible; if not, develop them where necessary.
- Keep a database of the system standards and the conforming commercial items.

For maintaining standards

- Participate in standards organizations so that the program's interests are represented.
- Expect and plan to change some (or all) of the standards during the system life cycle.

For selecting among contractors

- Look for evidence of previous contractor success with open systems in general and the chosen standards in particular.

For planning budgets

- Plan for additional systems engineering effort beyond what would normally be estimated.
- Do not base the budget or schedule plans on previous systems that *did not* use open systems.
- Budget for a testing capability that can be continuously employed.

For choosing between strategies

- Document all extensions to the standard that are used—it will make later replacement easier.
- Persist with a given standard for as long as it makes sense to do so, but be prepared for changes. If the commercial sector discontinues a standard, consider why it is (or is not) in the best interest of the program to continue to require it.

DANGER SIGNS

- Industry drops support for the standard after it is adopted by the program
- The standard changes substantially in a new revision and the program had no advance knowledge that this was going to happen
- Interoperability testing is downplayed because the interfaces "should work," since they are standards-based
- De-emphasizing conformance testing because the product implementing the standard has been "certified"
- During evaluation, only one commercial product that conforms to the standard that COTS products must meet is available. This likely indicates a flawed standards selection process.

IMPLEMENTATION RESOURCES

Meyers, B. Craig & Oberndorf, Patricia. *Managing Software Acquisition: Open Systems and COTS Products.* Boston, MA: Addison Wesley, 2001 (ISBN 201704544).

Place, Patrick R. H. *Guidance on Commercial-Based and Open Systems for Program Managers* (CMU/SEI-2001-SR-008, ADA392367). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/publications/documents/01.reports/01sr008.html

GUIDELINE 2: Commercial Off-The-Shelf (COTS)-Based Systems

Contributor: Dynamic Systems Program

2

GUIDELINE

Selecting a COTS-based development approach means the program office must assess the maturity of the commercial technology relative to the program's needs, reconcile the processes assumed by the COTS tool with the existing user processes, and manage the continuous evolution of the COTS technology and products.

SCENARIO

When replacing a custom-developed system, an organization's objective was to meet all of the collected needs of many different user sites, but still allow each site to follow its own customized business processes. The approach was to create a set of customized extensions to a few COTS products that they would integrate using custom-built code. What happened was

- The initial requirements did not focus on how the COTS products worked, but only on the system's requirements—until program management realized that the requirements had to address the end-to-end operation across the products.
- Integration was more difficult than anticipated, with most system defects being introduced in the code used to integrate the different COTS products. Many user-requested changes required modification of the COTS products and in each instance there were unpleasant side effects that the contractor did not anticipate.
- By creating custom extensions to the COTS products, instead of changing the existing business processes to fit the existing COTS product paradigms, development costs soared. The organization also assumed responsibility for maintaining each of these extensions, negating some of the benefits of choosing a COTS solution in the first place.
- The COTS products evolved during the three-year development effort. New capabilities, which in some cases obviated the need for some of the custom extensions, were added. These new features were not anticipated, and so could not be used in the final system.

ANALYSIS

While increasingly popular, COTS-based development presents various issues for acquisition:

- Assumptions about end-user processes are built into COTS products; these assumptions may not match your desired usage
- Addressing mismatches with COTS product licensing, data rights, and warranties
- The frequent, continuous evolution of COTS products and technology is driven by the market and acquirers have limited control over change
- COTS products can be difficult to integrate due to varying architectural paradigms across products, dependencies among those products, and limited visibility into their internal structures and behavior

A COTS-based acquisition approach demands a long-term commitment, as well as long-term acquisition, engineering, and business strategies, since expectations of short-term payoffs may be unrealistic. The potential benefits of using COTS components, such as lower construction and sustainment costs, faster development, or the ability to leverage new technology and the commercial marketplace, can require an early investment of both money and effort. Creating a system architecture that can evolve as the COTS products evolve and mature can require a greater expenditure in the short term, but can pay off in the medium to long term. If the program fails to keep its investment in a COTS-based system current and falls too far behind in product upgrades, the program may not be able to recover. Thus the acquisition strategy, contract vehicles, and stability of funding must be flexible enough to support the program for the long term.

In conclusion, you cannot assume that, as a routine acquisition cost-reduction strategy, incorporating COTS components is advantageous. There must be a reasoned judgment—a business case analysis—made about the applicability of COTS components to a given system.

REQUIRED ACTIONS

- Ensure that significant aspects of the system are *not* unprecedented in the marketplace and that combinations of COTS products have been successfully fielded for the application area (both in general and by the contractor).
- Design your system architecture to insulate the COTS components from other parts of the system so that those components can be replaced without affecting the entire system.
- Make sure the program is able to make ongoing, iterative trade-offs to find the best fit between engineering (architecture), business (marketplace), and contractual (system context) perspectives.
- Only adapt COTS products in ways the vendor supports (i.e., through scripting, wrappers or plug-ins/extensions), rather than modifying source code.
- Use pilot projects to validate the appropriateness of a given COTS component.
- Ensure that COTS product evaluations consider the diversity of products offered, the market share of those products, and the maturity of both the vendor and the products.
- Obtain the most current version of a COTS product and its documentation when conducting a product evaluation and ensure that the information is *still* current when the selection is made.
- Ensure that the program can afford the up-front costs (such as designing an evolvable architecture, or integrating COTS upgrades) needed to realize the long-term benefits of a COTS-based approach—and balance those against the risks incurred by *not* investing in COTS.
- Have the expertise and budgeted funding to regularly conduct COTS product/technology evaluations, integrate periodic product upgrades (depending on the rate of product change), and replace a COTS product when it is discontinued or the vendor goes out of business.
- Select COTS license types (enterprise-wide, per-seat, per user, and end-user/run-time) that fit your program structure. An inappropriate license type can drive up costs unnecessarily.
- Partner with COTS product vendors and participate in user groups to achieve insight into vendor plans and motivators, but do not expect to have influence over a COTS vendor to the point that you can change the direction of their product.
- Structure your program to be able to support continuous change throughout both development and sustainment as the COTS market evolves.

- Hire or train a development staff that will be able to conduct an integration effort, rather than a new development effort.
- Train developers on the COTS products that will be integrated into the system.
- Ensure that the PMO staff has the necessary experience and/or training in product licensing and technical product evaluation to either conduct or review COTS selection activities.

DANGER SIGNS

- PMO staff that is inexperienced or untrained in COTS product acquisition issues is given responsibility to select, or review the selection of, COTS products
- No budget is allocated for COTS product evaluations and other COTS-related activities beyond the initial ones

IMPLEMENTATION RESOURCES

Albert, Cecilia & Brownsword, Lisa. *Evolutionary Process for Integrating COTS-Based Systems (EPIC): An Overview* (CMU/SEI-2002-TR-009, ADA405844). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
http://www.sei.cmu.edu/publications/documents/02.reports/02tr009.html

Meyers, Craig B. & Oberndorf, Patricia. *Managing Software Acquisition: Open Systems and COTS Products.* Boston, MA: Addison Wesley 2001 (ISBN 0201704544).
http://www.awprofessional.com/bookstore/product.asp?isbn=0201704544&redir=1

Place, Patrick R. H. *Guidance on Commercial-Based and Open Systems for Program Managers* (CMU/SEI-2001-SR-008, ADA392367). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
http://www.sei.cmu.edu/publications/documents/01.reports/01sr008.html

SEI COTS-Based Systems Lessons Learned Web Pages
http://www.sei.cmu.edu/cbs/lessons/index.htm

SEI Education and Training: *COTS-Based Systems for Program Managers.* For more information about this course, visit http://www.sei.cmu.edu/products/courses/cots-progmgrs.html.

GUIDELINE 3: Software Architecture

Contributor: Anthony J. Lattanze

3

GUIDELINE

Developing a successful software architecture requires defining the quality attributes that drive the architecture, designating a chief architect, documenting the architecture with all views needed to communicate with stakeholders, and regularly evaluating the architecture as it is developed.

SCENARIO

A large acquisition program was assigned the challenging task of developing a system having a common computing and network infrastructure. Program management understood that designing a quality software architecture would be critical to the success of the large, integrated system.

Early in the acquisition process, the program began to manage the software architecture development process by focusing on software architecture support for system quality requirements. They used business drivers and stakeholder input to identify specific software architecture system quality needs such as reliability and maintainability. The group of diverse stakeholders often expressed conflicting system quality needs, so before and after the contract was awarded, the program office hosted SEI Quality Attribute Workshops[6] (QAWs) to clarify these needs. In addition, the program office provided SEI software architecture training for program office and contractor personnel to improve the software architecture capability across the team.

After the contract was awarded, the contractor began developing the software while the program office continued to focus on the software architecture by analyzing how the software architecture supported specific system qualities such as reliability. They used an SEI-developed scenario analysis approach called the Architecture Tradeoff Analysis Method® (ATAM®)[7] which helped identify software architecture trade-offs and risks and facilitated the gathering of input from many stakeholders.

The program office is realizing the benefits of their early and ongoing focus on software architecture, such as avoiding typical but costly software architecture mistakes, increasing confidence in their technical and architectural path, and better positioning the program for long-term sustainment. The focus on software architecture continues through development by obtaining stakeholder input to system quality needs, documenting the software architecture, maturing software architecture support for specific system quality concerns, and identifying software architecture tradeoffs and risks.

6 The Quality Attribute Workshop (QAW) is an SEI-developed method for analyzing a conceptual or system architecture against critical quality attributes before the software architecture is fully developed.

7 Architecture Tradeoff Analysis Method and ATAM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. The Architecture Tradeoff Analysis Method (ATAM) is an SEI-developed method for evaluating software architectures relative to quality attribute goals, exposing architectural risks that potentially inhibit the achievement of an organization's business goals.

ANALYSIS

System architectures define the context in which the parts or elements of the system will be designed and developed (or acquired). *Software* architectures perform the same function for the software elements, but both software and system architectures are essential for ensuring that the resulting system is fit for the intended purpose.

Some of the most critical aspects to developing successful software architectures are

- *Quality attributes*: These express the required "attributes of the system" and are often called the "-ilities" (reliability, scalability, maintainability, performance, availability, security, modifiability, interoperability, testability, etc.) Quality attributes are derived from the business and mission needs for the system and are used to drive the architecture design. They must be specific, quantifiable, and ideally defined by scenarios that illustrate the system's desired quality attributes. The SEI's QAW provides one method to identify and use these quality attributes for software architecture design.
- *Organizational issues*: There needs to be a *single* person who is the responsible chief software architect, even when there is an architecture team. Without a responsible architect reaching consensus is difficult, and designs may not stabilize. After the architecture is established, the organization should be structured to map to the major system components defined by the architecture and system teams (working independently, possibly from separate contractors) work on those system components.
- *Architecture documentation*: The software architecture and the documentation that describes it is the central artifact within the development life cycle. The primary deliverable from software architecture design is a set of different types of views created by the architect (or team) that, when viewed together, provide the information needed to support design analysis and architecture evaluation.
- *Architecture evaluation*: Software architectures must be regularly evaluated in a disciplined and repeatable way, with broad stakeholder involvement. The evaluation context should include the quality attributes and business/mission goals. Architecture evaluation can be done through simulation of the architecture, questions and checklists, scenario analysis, or by using techniques such as the SEI's ATAM. This allows the same stakeholder scenarios that were used to design the system architecture to be used to evaluate the software architecture, and identify the risks involved in achieving the system's desired quality attributes.

REQUIRED ACTIONS

- Ensure that the software requirements and software architecture teams work closely together to capture the architectural assumptions as the requirements are identified. The defining of requirements and architecture are closely linked, so both teams must work together to have a consistent and complete understanding of the requirements.
- Have stakeholders discover, document, and prioritize quality attributes as early as possible in the life cycle: 1) capture the concerns of all of the stakeholders, 2) allocate enough time and resources to do a thorough job, and 3) ensure that staff are properly trained in the definition of quality attributes.
- Make the software architecture documentation a formal deliverable and allocate sufficient resources for its creation and review. Ensure that it is complete, unambiguous, and widely available, includes multiple views, and is reviewed and updated periodically to stay synchronized with the system.

- Regularly evaluate the software architecture to determine its fitness with respect to business goals by including evaluations in your Request for Proposal (RFP) and contract, ensuring the software architecture meets both functional and quality attribute requirements, and using the original key scenarios to analyze the completed architecture. Schedule these evaluations, budget for them, and involve all stakeholders in them to manage any architectural risks identified during the evaluation.
- Designate a single, accountable chief software architect to architect the software or to head the architecture team. Structure the project teams to mirror the architecture and ensure adequate communication between teams designing and building system elements.

DANGER SIGNS

- Quality attributes for the system have not been identified or adequately defined ("The system shall be modifiable" or "The system must be secure" are not adequate definitions)
- The entire architecture is documented using a single view or a single view type
- Architectural documentation is inconsistent from one diagram to another
- Architectural documentation is substantially out of date
- The software architecture has not been evaluated, or has not been evaluated in the context of the original quality attributes and the business and mission goals
- Not all stakeholders were involved in the evaluation of the software architecture
- The software architecture team membership undergoes significant turnover throughout its lifetime or is disbanded after the architecture is initially defined
- The project's team structure does not clearly map to the architecture

IMPLEMENTATION RESOURCES

Barbacci, Mario R. *SEI Architecture Analysis Techniques and When to Use Them* (CMU/SEI-2002-TN-005, ADA413696). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/02.reports/02tn005.html

Bergey, John K. & Clements, Paul C. *Software Architecture in DoD Acquisition: A Reference Standard for a Software Architecture Document* (CMU/SEI-2005-TN-020). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn020.html

Bergey, John K. & Fisher, Matthew J. *Use of the Architecture Tradeoff Analysis Method (ATAM) in the Acquisition of Software-Intensive Systems* (CMU/SEI-2001-TN-009, ADA396096). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. http://www.sei.cmu.edu/publications/documents/01.reports/01tn009.html

Bergey, John K. & Wood, William G. *Use of Quality Attribute Workshops (QAWs) in Source Selection for a DoD System Acquisition: A Case Study* (CMU/SEI-2002-TN-013, ADA405848). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002. http://www.sei.cmu.edu/publications/documents/02.reports/02tn013.html

Clements, Paul; Bachmann, Felix; Bass, Len; Garlan, David; Ivers, James; Little, Reed; Nord, Robert; & Stafford, Judith. *Documenting Software Architectures: Views and Beyond.* Boston, MA: Addison Wesley, 2002 (ISBN 0201703726).

SEI Education and Training: *Software Acquisition Survival Skills.* Software Architecture Module. For more information about this course, visit http://www.sei.cmu.edu/products/courses/sass.html.

GUIDELINE 4: Software Sustainment

Contributor: William E. Novak

4

GUIDELINE

Key considerations in selecting organic sustainment vs. Contractor Logistics Support include the software complexity, the experience of the sustainment organization with the system type and technology, and any plans to continue further development after the system enters sustainment.

SCENARIO

A communications system acquisition was nearing completion of a major upgrade of the system to use a new underlying technology (Phase 2), which made extensive use of COTS components. The initial version of the system had been moved into sustainment, which was being performed by the contractor who had developed both the original system (Phase 1) and the upgraded system (Phase 2). Even as Phase 2 was being completed, the requirements for Phase 3, a second upgrade, were approved. A review was conducted to determine if both the system and the sustainment organization were ready to make the transition to sustainment. The review found deficiencies in both areas, determining that the program office had neither funded nor performed sufficient sustainment planning. While their default intention was to pursue organic sustainment at another site, rather than commercial sustainment (using Contractor Logistics Support, or CLS), the program office still had not produced adequate documentation or training for the system to be successfully transitioned to a different organization for sustainment. In addition, the identified sustainment organization did not have sufficient expertise in either COTS-based systems or the required communications technology to take on sustainment responsibilities without substantial additional training. The program office decided to delay the transition to sustainment by a year to complete their sustainment transition planning and to create the necessary documentation and training. The sustainment organization decided to contract with the original developer to perform the sustainment for an initial two-year period during which the sustainment organization could come up to speed on both the system and on COTS technology issues. However, these delays could have been avoided if explicit planning had been done well in advance with respect to how sustainment for the system would be performed.

ANALYSIS

A good working definition of software sustainment is, "The processes, procedures, people, materiel, and information required to support, maintain, and operate the software aspects of a system." Software sustainment encompasses all aspects of supporting, maintaining, and operating the software aspects of a system. It is a superset of software maintenance activities, which include corrective, adaptive, preventative, and perfective modifications. Software sustainment also addresses other issues not typically included in maintenance such as operations, documentation, deployment, security, configuration management, user and operator/administrator training, and user support.[8]

8 Lapham, Mary Ann. *Sustaining Software-Intensive Systems*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, to be published.

The scenario above illustrates the results of neglecting some of the key issues required in preparing a system for the transition to sustainment. Much of the preparation must be planned and executed from the beginning of the acquisition, so that the essential activities have been completed and the required documents are in place when it is time for the transition.

At a minimum, the acquisition strategy should identify whether the source of sustainment support will be commercial (CLS) or organic (government). Choosing between these alternatives requires evaluating both the system's software complexity and the ability of the sustainment organization to manage it. In addition, the strategy should take into account the sustainment organization's experience working with analogous kinds of systems and any plans for further development after sustainment begins, which could create significant configuration management challenges for simultaneously maintaining and developing two different baselines of the system.

REQUIRED ACTIONS

For general sustainment:

- Ensure that the government has rights to all data (including intellectual property) that will be needed to sustain the system.
- Make escrow provisions for all custom source code, documentation, and development tools used to build the system, and all COTS software products' source code, documentation, and development tools used to build them.
- Define the required levels of responsiveness, maintenance, and support for the system during sustainment and the funding required to achieve them (which may change over time).

For an organic (or different contractor) sustainment approach:

- Verify that the sustainment organization has experience with both the system domain and key aspects of the system. For example managing vendor relationships and transitioning licenses for a COTS-based system or sustaining a system requiring substantial interoperability with other systems can require additional expertise.
- Transition the needed COTS licenses from the development to the sustainment organization.
- Ensure that all data on the system (code, documentation, training materials, development tools, support tools, metrics, standards and guidelines, coding guidelines, etc.) and development data (Software Trouble Reports, bug report histories, enhancement requests, etc.) are maintained in systems that the government and the sustainment organization can both access can feasibly obtain, or require the developer to provide the data in a format that the government and sustainment organizations can use in their systems.
- Make system development experts available to work with the sustainment organization to help bring them up to speed.
- Provide adequate funding to the sustainment transition planning effort to bring the sustainment organization up to speed.
- Obtain a signed Source of Repair Assignment Process (SORAP) well before the sustainment contract is scheduled to be let.
- Implement strong configuration management processes to synchronize the system's sustainment baseline with any continuing development baseline.

- Manage development staff expertise by
  - o offering a compelling retention plan to encourage key staff to remain with the effort until the end of development
  - o training the sustainment organization early to minimize the effects of any developer expertise loss that does occur
- Develop and provide formal training for system users, administrators, operators, developers, and maintainers.
- Plan and fund user support functions such as a Help Desk. Measure this level of support (quality, responsiveness, etc.) to ensure that it is adequate.
- Ensure that clear Information Assurance (IA) requirements exist and that there is a planned path for the transition of IA to the sustainment organization.
- Ensure that the system is developed without an over-reliance on the developer's development processes and proprietary tools and that there is a plan in place (or in development) for the transition to the sustainment organization.

DANGER SIGNS

- A spiral development effort using CLS by the original development contractor is extended indefinitely because a government organization suited for conducting organic sustainment is not available, funded, or trained for assuming this responsibility
- Sustainment planning is not considered or is postponed during the early phases of the program
- Discussions with the selected or mandated organic sustainment organization reveal that they have little or no experience in the domain space or in software technology
- A signed SORAP is not in place in sufficient time to be able to let the RFP for the sustainment contract or to receive and evaluate proposals prior to the planned contract start date

IMPLEMENTATION RESOURCES

Acquisition Community Connection (ACC). *Sustainment Planning Resources.* https://acc.dau.mil/simplify/ev.php?ID=59264_201&ID2=DO_TOPIC (2004).

Bergey, John; Smith, Dennis; & Weiderman, Nelson. *DoD Legacy System Migration Guidelines* (CMU/SEI-99-TN-013, ADA370621). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
http://www.sei.cmu.edu/publications/documents/99.reports/99tn013/99tn013abstract.html

Cohen, Sholom; Dunn, Ed; & Soule, Albert. *Successful Product Line Development and Sustainment: A DoD Case Study* (CMU/SEI-2002-TN-018, ADA407788). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2002.
http://www.sei.cmu.edu/publications/documents/02.reports/02tn018.html

Seacord, Robert; Plakosh, Daniel; & Lewis, Grace A. *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices.* Boston, MA: Addison-Wesley, 2003 (ISBN 0321118847).

GUIDELINE 5: Requirements Development

Contributor: Software Engineering Process Management Program

5

GUIDELINE

The process of developing software requirements consists of designating a requirements team, defining a process for specifying requirements, applying use cases and scenarios to analyze the requirements, and involving stakeholders to validate the requirements—and doing so throughout development.

SCENARIO

In one program charged with defining a new COTS-based management information system (MIS), the process used to develop the software requirements did not capture the existing business processes versus the desired business processes or create a mapping between them. The "mapping" approach had been discouraged because the field office sites each used different business processes with the existing system. Instead, employees from the field sites were asked to submit their requirements for the new system. The number of requirements reached over 2,000, with some being very specific, while others were so basic or general that they were unusable. As a result, the requirements were not very useful to the contractor in defining the system.

A key issue was that the requirements did not address the end-to-end operational processes across the system and consequently, were not testable. Since testing was based on the requirements, executing a test script could prove that a given function worked, but not that a complete process could be performed from start to finish.

The contractor convinced the PMO to discard the requirements-based approach, pointing out that while the deliverable might meet the requirements, the program might not have a workable system. Ultimately, the program wasted almost a full year of effort before changing their focus to the business processes.

ANALYSIS

The program in the scenario initially adopted an approach to developing requirements that did not allow them to validate full system operation, as defining use cases and operational scenarios could have. The Capability Maturity Model® Integration (CMMI®) defines a process framework for developing requirements that identifies three high-level steps to be followed

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

- *Develop customer requirements*: This activity involves collecting stakeholders' needs, gathering stakeholders' requirements (along with their associated evaluation criteria), and then developing the customer requirements by documenting the analysis of the requirements. Use existing software assets as a source of information for understanding and defining new software requirements.
- *Develop product requirements*: The next activity in requirements development is most commonly performed by the contractor and will establish product and product-component requirements, allocate the requirements to the system components (analyzing existing software assets for reuse opportunities), and identify any interface requirements to other systems.
- *Analyze and validate requirements*: The third activity establishes operational concepts and scenarios for the system, defines what constitutes "required" functionality (i.e., what standard does it have to meet?), analyzes the requirements to achieve balance (i.e., balance stakeholder needs and constraints with respect to such considerations as cost, schedule, performance, functionality, reuse, maintainability, or risk), and validates the requirements (i.e., obtain agreement that the system requirements satisfy the stakeholders' requirements). Both the customer requirements and the product requirements need to be analyzed and specified.

Keep all of the system stakeholders as directly involved in the requirements development and management process as possible to help prevent or mitigate any miscommunications that may arise during the negotiation of requirements trade-offs.

One key to successfully developing requirements is to create a robust set of use cases to specify the functional requirements, concentrating first on use cases that typify the system's intended usage. Similarly, the use cases can be used to drive "storyboards" for specifying the system's user interfaces, and can also be used to define system acceptance criteria. Even components without a user interface (a "black box") can be treated as though they have one by asking what visibility or data is needed to determine if the box functions properly.

Developing "quality attribute"[10] scenarios is another requirements development technique used to specify quality requirements in a way that facilitates later evaluation as to whether the software architecture can meet those requirements. Quality attributes are identified for the system based on the key needs that drive the program from both a mission and a business perspective—be it performance, security, reliability, interoperability, or other attributes. Quality attributes must be clearly and quantifiably specified.

Requirements development also plays a role in the acquisition approach a program chooses. For example, if all customer requirements for an unprecedented system are defined up-front in a single-step acquisition approach and given to the contractor to implement, this early "freezing" of the requirements can result in a system that is obsolete by the time it is delivered. Furthermore, by defining all of the requirements before development begins, requirements improvements that might become possible due to new technology or a better understanding of the system are precluded. An evolutionary acquisition approach would allow these requirements to be expanded and developed iteratively.

10 Quality attributes, as described in the Software Architecture guideline, express system attributes such as reliability, scalability, maintainability, performance, availability, etc.

In general, develop software requirements through an iterative process at both the customer (system) and product (component) levels. Start with a high-level description of the functionality that needs to be provided and refine it as more is understood about the possible implementation alternatives. Iterate until the identified software requirements meet your criteria, which should include being complete, consistent, unambiguous, verifiable, documented, and testable.

REQUIRED ACTIONS

- Create a team that is responsible for performing requirements development (and management) activities, designate a leader, and ensure that the members are properly experienced or trained.
- Make sure that the PMO has enough resources to properly handle the PMO requirements work, and ensure that the same is true for the contractor.
- Develop (and then manage) requirements within the context of a complete and defined *process* so that all requirements are handled consistently.
- Put in place a written policy for establishing and managing software-related contractual requirements.
- Focus on identifying and fixing requirements problems early—not doing so can consume 25–40% of the budget.
- Use multiple elicitation and analysis techniques, rather than a single approach. This applies even to use case modeling, because it is best suited for developing functional requirements and less appropriate for other types of requirements.
- The PMO and the contractor should explicitly negotiate both the necessity of each requirement ("necessary" versus "nice to have") and their explicit priorities, especially where COTS products are involved.

DANGER SIGNS

- *Requirements are vague and subject to multiple interpretations*: This may be because the requirements were not documented clearly, or the contractor may be having difficulty in decomposing the government requirements, or the people developing the requirements have not been trained in the proper forms used to express requirements clearly.
- *Missing or extraneous requirements*: The PMO may have missed some stakeholders or types of requirements (such as quality attributes) or stakeholders may have specified desirable rather than needed features.
- *Dissatisfied stakeholders*: Dissatisfaction can be caused by a lack of access to the true stakeholders or poor representation of their needs. This may become apparent later in the project, but keeping stakeholders involved throughout the project may avoid, or alleviate such dissatisfaction earlier.
- *Unresolved conflicts between requirements*: Conflicting requirements need to be caught early. Example: the system must have a high degree of security, but also minimum log-in requirements.
- *Test cases are hard to develop; software is difficult to test*: Considering verification issues early makes this easier to correct than it will be later in the program.

- *Interface requirements are not defined and documented; interface agreements are not established and documented*: Occurs when a stakeholder is forgotten, along with the interfaces that impact them.
- *Overemphasis on functional requirements*: This is easy to spot early, and is apparent in missing or poorly specified data, interface, and quality requirements, and constraints.
- *Overemphasis on "sunny day" (i.e., normal path) scenarios*: Focusing primarily on how the system operates under normal circumstances may indicate that the potentially larger set of "rainy day" (exceptional path) requirements has been missed.

IMPLEMENTATION RESOURCES

SEI Education and Training: *Software Acquisition Survival Skills*. It All Begins With Requirements Module. For more information about this course, visit http://www.sei.cmu.edu/products/courses/sass.html

Software Engineering Institute, Carnegie Mellon University. CMMI-SE/SW/IPPD V1.1 Continuous Representation. 2002. Pittsburgh, PA. http://www.sei.cmu.edu/cmmi/models

Woody, Carol. *Eliciting and Analyzing Quality Requirements: Management Influences on Software Quality Requirements* (CMU/SEI-2005-TN-010). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn010.html

GUIDELINE 6: Requirements Management

Contributor: Software Engineering Process Management Program

6

GUIDELINE

Software requirements management requires having a clear baseline, change control, bi-directional traceability to work products, metrics on stability, and continuous stakeholder involvement.

SCENARIO

One program was upgrading two legacy financial systems and simultaneously integrating them into a single system. A requirements baseline was not established for the program, either before or during the upgrade and integration. While the contractor attempted to establish a baseline set of requirements and submitted it to the PMO, the PMO would not sign off on it. The PMO's explanation was that since it was a small office, they did not have sufficient staff or expertise to review or approve it. The resulting lack of a formal requirements baseline led to a number of different problems for the program.

One problem was that the requirements were volatile. Since they were not fixed by a baseline, every time the prototype system was shown or demonstrated, new requirements in the form of suggestions were added to the program.

Another problem was contention between the PMO and the contractor about whether adding particular capabilities not yet present in the system were considered enhancements or corrections of deficiencies. If adding the capability qualified as an enhancement, the PMO would have to pay for it; if it were deemed to be a deficiency, the contractor had to fix it at their expense. Unfortunately, the only way to determine the difference between an enhancement and a discrepancy is to have a firm requirements baseline to compare the capability against, which did not exist. This caused strained relations between the PMO and the contractor, and ultimately the program delivered only 75% of the desired capability to its users.

ANALYSIS

The lack of a requirements baseline had a significant impact on the program described in the scenario. The creation and maintenance of a requirements baseline is the focus of requirements management. The CMMI process framework for managing requirements identifies the high-level steps to be followed for this area. These steps are listed here, along with some guidance in each area.

- *Obtain an understanding of requirements:* Requirements can be managed more efficiently throughout development if they are categorized by their relevant attributes early on. For example, distinguish between software requirements that are critical to performance and those that can be easily traded, or between firm, well-understood requirements and those that only represent an initial understanding (and will need further investigation). Also, distinguish between requirements that must be implemented in software and those for which an alternate solution may be available—so that requirements may be moved if the architecture demands it (for example, mechanical vs. electronic control systems in an automobile's braking system).

- *Obtain commitment to requirements from all stakeholders*: Keep stakeholders informed on a regular basis about the status and disposition of their requirements. This provides regular opportunities for communication, and will help prevent simmering requirements issues from becoming significant problems unexpectedly.
- *Manage requirements changes*: Develop, baseline, and maintain critical requirements at the program level and place them under change control early in the program. Categorize the requirements as being "risky," "non-risky," or "unknown," and capture the assumptions that led to those categorizations.
- *Maintain bi-directional traceability between requirements and work products*: Maintain bi-directional traceability between the customer requirements, the product requirements and the work products to be delivered throughout the development effort. This is critical for estimating the impact of both requirements and architectural changes, as well as for ensuring full test coverage, and requires automated tool support in larger programs.
- *Identify inconsistencies between work products and requirements*: Analyze all changes to requirements for any impact a change may have on performance, architecture, supportability, system resource utilization, evaluation requirements, and contract schedule and cost.

The government PMO is generally responsible for the customer (or system) requirements, while the contractor is responsible for the derived product requirements, which include the software requirements. However, the government needs to be concerned both with managing (and developing) the customer requirements within the PMO, *and* with monitoring the contractor's product requirements management (and development) activities. The PMO should give these two tasks equal importance. To take this one step further, the PMO also has oversight responsibility for how requirements are decomposed and allocated down to the subcontractors.

The PMO and contractor must agree on the priority and allocation of requirements to the planned builds of the system. There should be a clear policy that is consistently followed in this allocation. For example, there needs to be a conscious decision on whether to implement the highest *risk* requirements first (to "burn down" program risk as early as possible) or to implement the highest *value* requirements first (to demonstrate important progress to stakeholders).

REQUIRED ACTIONS

- Place software-related contractual requirements under change control prior to the release of the solicitation package.
- Establish a pre-contract requirements baseline.
- Establish measurements to determine the effectiveness of the requirements management (and development) activities, including requirements stability and volatility. These may include the number of new requirements, the number of changed requirements, the complexity involved, and the effort needed.
- Use automated requirements tools and a requirements repository for tracking and tracing large numbers of requirements. Ensure that the requirements tools used by the PMO and the contractor are identical and that there is compatibility across the toolsets used for different activities within the PMO.
- Ensure that the government has the rights to the derived product requirements that are maintained by the contractor.
- Involve *all* stakeholders throughout the program in the requirements process—and obtain their sign-off.

DANGER SIGNS

- *Inability to verify that the design or architecture meets requirements or stakeholder needs*: Early system testing may catch these problems and limit required rework, but recovery may be difficult and costly.
- *Missing documentation*: If the requirements documentation is being manually generated, check the status of the documentation early and often, and ensure that it is updated before system delivery.
- *Lack of baseline or numerous baselines*: The lack of a requirements baseline is unacceptable; having numerous baselines create confusion.
- *Requirements volatility and/or growth ("scope creep")*: Volatile requirements and requirements growth can result in resource shortages and cost overruns, and ultimately schedule slips, all due to rework. While some degree of volatility and growth are almost inevitable on most programs, estimating the amount and rate of change up-front, and then closely monitoring the actual change will allow this to be understood and controlled.
- *Requirements shifting from earlier to later increments of an evolutionary acquisition*: This movement may indicate underlying resource shortages, or an underestimate of complexity. The result can be a cumulative "bow wave" effect with the most difficult development being postponed until the end of development, when there may not be time to implement it.
- *Solution does not match stakeholder needs*: Keep stakeholders involved and develop early prototypes to identify such requirements and deliverable mismatches early. This should occur well before system validation, while the cost of making changes is still comparatively low.

IMPLEMENTATION RESOURCES

SEI Education and Training: *Software Acquisition Survival Skills*.
It All Begins With Requirements Module. For more information about this course, visit
http://www.sei.cmu.edu/products/courses/sass.html.

Software Engineering Institute, Carnegie Mellon University.
CMMI-SE/SW/IPPD V1.1 Continuous Representation. 2002. Pittsburgh, PA.
http://www.sei.cmu.edu/cmmi/models

GUIDELINE 7: Operational Information Assurance

Contributor: Carol Woody, PhD

7

GUIDELINE

The introduction of a new system into an operational environment can include extensive operational changes. Existing operational procedures cannot be assumed to be sufficient. A risk evaluation of the targeted operational system is critical for maintaining effective information assurance when a new system is introduced.

SCENARIO

The contract for the development of a major federal agency system specified that current operational security requirements must be applied to the new system and the contractor was given a copy of the established operational procedures. For more than twenty years, this process was standard procedure for introducing a new system into the existing environment. However, in this case, the development effort constituted a major modernization of the existing environment. The existing operational procedures hampered and in some cases, hurt the development process and did not take into consideration the information assurance needs of the target environment. Validation of delivered information assurance was limited to penetration testing on the final production environment. Testing also did not consider the increased risk inherent in the move from a mainframe environment to a distributed UNIX environment.

A risk evaluation was performed on the developed system but only risks within the system were considered. No evaluation was performed to identify the risk to existing operations caused by introducing an entirely new operating environment into the existing one.

ANALYSIS

The implementation of a new system represents risk to both the new development and the current operational environment. Existing operational mechanisms used to address risk are not always applicable, especially when the technology platform changes, COTS components are used, and integration is expanded. Each project must be evaluated to determine the risk of information assurance failure with regard to the mission of the organization.

A risk assessment must be part of each major development milestone and the assessment must consider the target operational environment, which may differ drastically from the current one.

REQUIRED ACTIONS

Establish appropriate contracting mechanisms to ensure that operational risk is considered during development

- Establish information assurance requirements as an integral part of the development effort.

- Develop requirements for addressing operational risk based on
  - o an analysis of the impact of the system to the operational environment
  - o the Mission Assurance Category (MAC) level assigned to the system
- Consider appropriate operational risk and approval by the designated approving authority (DAA) with regard to system acceptance.

Required deliverables must be subject to information assurance review by independent verification and validation (IV&V)

- Include issuance of the Authority to Operate (ATO) as part of the system acceptance for contracted deliverables.
- Include operational mechanisms for maintaining the ATO in the implementation planning for a new system.

Incorporate clear quality targets for information assurance within the requirements

- Clearly establish the MAC.
- Clearly define steps for meeting information assurance requirements and establish the DAA.
- Allocate resources for assembling and validating the System Security Authorization Agreement (SSAA).

DANGER SIGNS

- Mission's information assurance needs are unclear or conflicting
- Steps for obtaining an ATO are not included in overall system planning
- Operational evaluation of a new system is limited to penetration testing of hardware platforms

IMPLEMENTATION RESOURCES

Alberts, Christopher; Dorofee, Audrey; & Woody, Carol. *Considering Operational Security Risks During System Development.* http://www.cert.org/archive/pdf/alberts-2.pdf (2002).

Defense Information Systems Agency (DISA). *2004 DISA Contracts Guide.* http://www.disa.mil/main/support/contracts/idiq_iassure.html (2004).

DoD. *Department of Defense Directive Information Assurance (IA) (DoD 8500.1).* http://www.dtic.mil/whs/directives/corres/pdf/d85001_102402/d85001p.pdf (2002).

DoD. *Department of Defense Instruction Information Assurance (IA) Implementation (DoDI 8500.2).* http://niap.nist.gov/cc-scheme/policy/dod/d85002p.pdf (2002).

Information Assurance Technology Analysis Center (IATAC). *Technical Area Tasks (TATS).* http://iac.dtic.mil/iatac/TATs.html (2005).

National Institute of Standards and Technology (NIST). *Guide for Security Certification and Accreditation of Federal Information Systems (NIST Special Publication 800-37).* http://csrc.nist.gov/publications/nistpubs/800-37/SP800-37-final.pdf (2004).

National Security Agency (NSA). *National Security Agency Central Security Service Frequently Asked Questions - IA.* http://www.nsa.gov/about/about00019.cfm (2005).

GUIDELINE 8: Information Assurance for COTS Sustainment

Contributor: Carol Woody, PhD

8

GUIDELINE

The transition to sustainment for systems containing COTS components must make special provisions to enable the sustainment organization to evaluate the impact of COTS component changes, independently assess the criticality of security flaws, and accommodate additional support for IA testing and validation.

SCENARIO

A system built using COTS components is scheduled to enter sustainment when a current iteration of spiral development is complete. At this time, the Authority to Operate (ATO) will be issued for three years; however, the typical refresh cycle on the COTS products within the system is less than two years. In addition, patches are periodically issued by the COTS vendors to address critical security problems in the software. After a patch becomes available, the potential of an attack using that flaw increases substantially for the systems that do not apply the patch.

Each time a COTS component is patched, a thorough revalidation of the system is required to ensure that the vendor's changes have not broken integration mechanisms implemented during system development. Should changes in the COTS components impact system functionality and performance in any way, skilled developers must be available to fix the system. For the sustaining organization to successfully support the COTS components, a fully functional development environment for inserting and validating changes must be available, in addition to a robust production-like test environment for performing revalidation tests.

ANALYSIS

In the scenario above, the sustainment organization may not be able to limit upgrades to a single annual configuration release, as they have in the past. Each COTS component is changing at a different rate based on vendor support decisions. The impact of each change must be evaluated against the mission that component performs in the fielded system. The sustainment organization must acquire sufficient knowledge of the fielded system to evaluate the impact.

In addition, the sustainment organization cannot rely on a vendor to determine the criticality of a security flaw. The vendor can provide a range of concern, but skilled security experts familiar with the use of the component within the fielded system must evaluate each flaw based on the risk to the mission of the system and the potential impact. The Defense Information System Agency (DISA) Information Assurance Vulnerability Management (IAVM)[11] can be used as a basis for evaluating components being monitored by DISA.

11 The Defense Information System Agency (DISA) Information Assurance Vulnerability Management (IAVM), Joint Task Force-Global Network Operations (JTF-GNO) Web site (http://www.cert.mil).

COTS components continuously change based on vendor support decisions and identified security flaws. Using these components requires the capability to adjust fielded systems with regular upgrades and corrections to maintain information assurance. This level of change during sustainment is greater than that of previously supported code. Meeting information assurance requirements during sustainment of COTS components also requires additional support for testing and validation.

REQUIRED ACTIONS

Provide the following information assurance artifacts at transition to sustainment:

- The MAC level, which is approved by the designated approving authority (DAA), is the basis for defining the level of information assurance required by a system. This designation is based on the requirements for confidentiality, critical system resources (CSR), and critical program information (CPI) available within the system.
- A Command, Control Communications, Computers, and Intelligence Support Plan (C4ISP)
- System Security Authorization Agreement (DITSCAP product), which may be established at the component level or at the overall functional level. Typically, these documents are developed for independent components that may function together to address a mission. An understanding of risk is needed for both the component and overall functional mission perspectives.
- A Security Policy Compliance Assessment (SPCA)
- A Program Protection Plan (not required unless MAC level indicates CRS or CPI)

Responsibilities of the sustainment organization are to

- Consider configuration management for COTS product upgrades
- Evaluate each proposed update to confirm that the system is maintaining a low risk posture with regard to information assurance
- Establish mechanisms to expeditiously validate and implement updates and upgrades for COTS products that impact information assurance
- Establish mechanisms for analyzing risk at the component level and overall integrated functional level:
  - o component-level risk can be addressed by using the DISA IAVM
  - o the Operationally Critical Threat, Asset, and Vulnerability Evaluation SM (OCTAVE ®) methodology provides a risk approach for the integrated functional level to link risk to the impact on the mission of the system

DANGER SIGNS

- No specific provisions in sustainment configuration management for information assurance
- No involvement of the designated approval authority in the transition to sustainment
- No SSAA acceptance review by the sustainment organization
- No provisions for risk management beyond the component level in sustainment

SM Operationally Critical Threat, Asset, and Vulnerability Evaluation is a service mark of Carnegie Mellon University.
® OCTAVE is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

IMPLEMENTATION RESOURCES

Common Criteria Evaluation and Validation Scheme (CCEVS).
http://niap.nist.gov/cc-scheme/vpl/vpl_type.html

DoD. *Department of Defense Instruction Information Technology Security Certification and Accreditation Process (DITSCAP) (DoDI 5200.40).* http://iase.disa.mil/ditscap/ (1997). Supplemented by DoD 8510.01-M, Applications Manual.
http://www.dtic.mil/whs/directives/corres/html/85001.htm (2000).

DoD. *Department of Defense Directive Information Assurance (IA) (DoD 8500.1).*
http://www.dtic.mil/whs/directives/corres/pdf/d85001_102402/d85001p.pdf (2002).

DoD. *Department of Defense Instruction Information Assurance (IA) Implementation (DoDI 8500.2).*
http://niap.nist.gov/cc-scheme/policy/dod/d85002p.pdf (2002).

DoD. *Defense Information Assurance Certification and Accreditation Process (DIACAP).* (Future replacement for DITSCAP based on DoD 8500.1 and 8500.2 controls; compliant with FISMA.

Federal Information Security Management Act (FISMA) Implementation Project.
http://csrc.nist.gov/sec-cert/

Information Assurance Support Environment (IASE). *IA Document Library.*
http://iase.disa.mil/stigs/iadocs.html

International Common Criteria for Information Technology Security Evaluation (basis for NIAP evaluation). http://www.commoncriteriaportal.org

National Information Assurance Partnership (NIAP). http://niap.nist.gov/

National Institute of Standards and Technology (NIST). *Guide for Security Certification and Accreditation of Federal Information Systems (NIST Special Publication 800-37).*
http://csrc.nist.gov/publications/nistpubs/800-37/SP800-37-final.pdf (2004).

National Institute of Standards and Technology (NIST). *Security Considerations in the Information System Development Life Cycle (NIST Special Publication 800-64).*
http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf (2004).

Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE)
http://www.cert.org/octave

GUIDELINE 9: Information Asset Protection

Contributor: Carol Woody, PhD

9

GUIDELINE

All information assets critical to the mission of the organization must be protected. Management cannot assume that appropriate protection is provided as a by-product of normal use. Protection must be planned, established, and monitored.

SCENARIO

A network administrator, just before being dismissed, set up a software time bomb that destroyed a group of applications and deleted databases a week later. Since the backup files were missing and no one had been assigned to assume responsibility for the file server and operating system, the problem was not corrected in a timely manner. Employees were sent home for several days, temporary employees were dismissed, and technicians scrambled to rebuild the data from paper documents and older electronic versions. Copies of important documents had to be retrieved from other organizations to rebuild the information resources. The work of the organization was seriously delayed and management was embarrassed to admit to such poor technical coverage of the organization's information assets.

ANALYSIS

Most organizational work relies heavily on the availability of information resources. It is critical that these assets are protected to ensure availability. Identification of critical assets and determination of the appropriate level of protection for them requires analysis and planning.

Information assets include all forms of electronic content that require control. It also covers important intellectual property, confidential and classified information, research data, and personal identifiable data that is sensitive but unclassified.

REQUIRED ACTIONS

Identify critical assets and protection needs

- Inventory information assets
- Classify current assets as to criticality and sensitivity.
- Establish a MAC and designated owner for each asset.
- Define appropriate protection requirements for each applicable classification category.

Analyze current protection efforts

- Review operational protection plans for completeness
- Verify that current operational plans are being applied

Analyze information security risks to determine sufficiency of existing protection efforts

- Perform an operational risk analysis
- Develop a strategic plan to align policy, procedures, and practices for effective information protection

DANGER SIGNS

- One individual has sole responsibility for major information assets
- Limited information classification and planning for asset protection
- Current efforts are assumed to be sufficient, but have not been validated

IMPLEMENTATION RESOURCES

Alberts, Christopher & Dorofee, Audrey. *Managing Information Security Risks: The OCTAVE Approach.* Boston, MA: Addison-Wesley, 2003 (ISBN: 0321118863).

Alberts, Christopher; Dorofee, Audrey; Stevens, James; & Woody, Carol. *Introduction to the OCTAVE Approach.* http://www.cert.org/octave/approach_intro.pdf (2003).

Allen, Julia. *Governing for Enterprise Security* (CMU/SEI-2005-TN-023). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
http://www.sei.cmu.edu/publications/documents/05.reports/05tn023.html

Caralli, Richard A. *The Critical Success Factor Method: Establishing a Foundation for Enterprise Security Management* (CMU/SEI-2004-TR-010, ESC-TR-2004-010) Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2004.
http://www.sei.cmu.edu/publications/documents/04.reports/04tr010.html

Cyber Security Industry Alliance (CSIA). *Ten Steps for Securing Electronic Health Care Systems.* https://www.csialliance.org/resources/pdfs/CSIA_Health_Care_April_2005.PDF (2005).

Ellison, Robert J.; Mead, Nancy R.; Longstaff, Thomas A.; & Linger, Richard C. "The Survivability Imperative: Protecting Critical Systems." *CrossTalk Vol. 13*, No. 10 (October 2000): pp. 12–15, 25.
http://www.stsc.hill.af.mil/crosstalk/2000/10/linger.html

Microsoft TechNet. *The Security Risk Management Guide.*
http://www.microsoft.com/technet/security/topics/policiesandprocedures/secrisk/default.mspx (2004).

National Institute of Standards and Technology (NIST) *Guide for Developing Security Plans for Information Technology Systems (NIST Special Publication 800-18).*
http://csrc.nist.gov/publications/nistpubs/800-18-Rev1/sp800-18-Rev1-final.pdf (1998).

National Institute of Standards and Technology (NIST). *Security Self-Assessment Guide for Information Technology Systems (NIST Special Publication 800-26).*
http://csrc.nist.gov/publications/nistpubs/800-26/sp800-26.pdf (2001).

National Institute of Standards and Technology (NIST). *Risk Management Guide for Information Technology Systems (NIST Special Publication 800-30).*
http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf (2002).

National Institute of Standards and Technology (NIST). *Contingency Planning Guide for Information Technology Systems (NIST Special Publication 800-34).*
http://csrc.nist.gov/publications/nistpubs/800-34/sp800-34.pdf (2002).

National Institute of Standards and Technology (NIST). *Volume I: Guide for Mapping Types of Information and Information Systems to Security Categories, Version 2.0 (NIST Special Publication 800-60).*
http://csrc.nist.gov/publications/nistpubs/800-60/SP800-60V1-final.pdf (2004)

GUIDELINE 10: Software Testing

Contributor: Julie B. Cohen

10

GUIDELINE

Successful software testing is tightly tied to the requirements, tests all software products needed by the system at the required levels, and addresses "non-functional" as well as functional requirements.

SCENARIO

A PMO was preparing to issue an RFP for the networking segment of a complex communications system. The program was aware that in a software-intensive system, especially one that would need to integrate with many different contractors, testing would be critical and could not be left until the end.

As a result, the overall test program became a focus area in the RFP. Bidders were asked to describe their approach to 1) leading edge-to-edge network integration, testing, and verification, 2) providing a comprehensive plan for segment-level testing, 3) supporting the testing done by the other segment contractors, and 4) supporting full system interoperability testing.

The RFP required bidders to describe the simulation and testbed facilities and tests for testing and validating the design, and hardware needs for design, requirements, verification, and test. They had to describe their planned support for the system engineering test and verification processes for the system, development of the system test and verification plan (including COTS product testing plans), and their plan to integrate and test the final implementation, including the system engineering methods and tools.

The PMO was aware that, due to the complexity of the system, they did not have all of the expertise they needed to fully specify their testing needs or to evaluate the proposals properly. They requested and received help from a government system integration group who acted as advisors on creating the RFP language and reviewed the testing portions of the proposals during source selection.

The result of this approach was that bidders were forced to be very detailed in their proposals regarding testing and the proposals could be evaluated on the credibility (cost, schedule, and performance) of the test program. The PMO did not expect any of the proposals to include a "perfect" test program, but the award could be at least partially based on the testing proposed and any areas of concern based on the proposal could be resolved much earlier in the program.

ANALYSIS

This scenario illustrates the extent to which a program can make testing a criterion for potential contractors when it poses a significant risk to the program's success. While testing may not be equally important in all programs, it requires an explicit strategy and cannot be treated as an afterthought.

The PMO must review the system test plans to ensure that software testing is adequately integrated into the overall test strategy. The PMO must also review software test plans for coverage and adequacy, enforce maintaining traceability from requirements to testing, and ensure that there is independent quality assurance from the start. It is also important to plan for visibility into software testing done by subcontractors. Specifically, plan to

- consider software test during source selection if it is a critical component of the system development
- test a variety of different types of software products
- review the levels and types of software testing needed and plan for those
- test "non-functional" requirements (performance, usability, etc.) as well as functional requirements
- make available experienced and trained PMO staff to witness all software testing
- involve the Operational Test team in test planning and test observation
- provide necessary support to ensure the government's Test and Evaluation Master Plan (TEMP) is developed on time and is updated throughout the program life cycle

Verification testing: Verification testing must be tightly coupled back to the system requirements—and even validation testing must be done with an understanding of what the system was intended and designed to do. If all stakeholders (including those conducting validation testing) are involved in the requirements of the program from the beginning, these expectations will be better understood and acknowledged.

Robustness testing: Operational Test and Evaluation (OT&E) is about testing the system and software "... under realistic operational conditions with users who represent those expected to operate and maintain the system when it is fielded or deployed."[14] Testing cannot focus solely on meeting derived contractual requirements, but must also validate that systems "...are effective and suitable for combat"—by testing under loaded conditions, testing for invalid user input, and other robustness issues that may occur under realistic operational conditions (stress testing, security, error detection/correction, graphical user interface (GUI) structure, external system interfaces, measurement unit conversions, timing and synchronization, memory allocation and usage, and path coverage). Also, involve OT&E staff in overall test planning during development. The earlier the contractor knows how the system will be tested operationally, the better chance they will build a system that can pass testing.

14 Army Regulation 73-1, Test and Evaluation Policy.

REQUIRED ACTIONS

Test all types of software products that are required for system success

- *System software*: The software that was contracted
- *User interfaces*: There may be different user interfaces for different installations
- *Modeling and simulation software*: This must be tested to determine the bounds of usefulness
- *Training software*: Simulators and other training materials, including on-line training compact discs (CDs)
- *Communications interfaces*: How does this system "talk" to other systems?
- *COTS products/interfaces*: Do the COTS products work, and do their interfaces work?
- *Programming interfaces (extensions)*: Test any extensible frameworks for that ability

Ensure that all types of applicable software testing are addressed in the software testing plan

- *Peer reviews*: Conduct peer reviews early and often
- *Unit testing*: Perform unit testing for each computer software configuration item (CSCI)
- *Integration testing*: Perform integration testing for multiple inter-related CSCIs
- *Interface testing*: Interface testing involves both internal and external system interfaces
- *Usability testing*: Usability testing ensures that the system is usable by your expected user
- *Verification testing*: Confirm that the system meets the specified requirements
- *Validation testing*: Confirm that the system is able to meet operational needs
- *Robustness testing*: A collection of testing approaches that demonstrate the system's ability to perform under exceptional circumstances.
- *System testing*: System testing provides end-to-end testing of the system functionality
- *Acceptance testing*: Determine what software tests need to be run for user acceptance
- *Interoperability testing*: Determines the ability of systems to provide data to other systems, receive data back, and use the data exchanged to operate effectively together.
- *Security testing*: A type of robustness testing that ensures that a system can restrict access to authorized personnel, and that the authorized personnel can only access the functions available to their security level
- *Fall-back testing*: Fall-back testing ensures functionality of failure modes, maintenance modes, etc.
- *Operational testing*: Operational testing is a key government responsibility
- *Regression testing*: Regressions testing verifies correctness of existing functionality in each new release

DANGER SIGNS

- Testers are told to execute *only* the test scripts, which always deliver the correct results (i.e., "happy path" testing)
- Fixes to known problems are not incorporated into the regression test suite
- All testing is conducted by the contractor that developed the system
- Tests are not traceable back to one or more system requirements
- Integration testing begins before all unit testing is completed

IMPLEMENTATION RESOURCES

Cohen, Julie; Plakosh, Dan; & Keeler, Kristi. *Robustness Testing of Software-Intensive Systems: Explanation and Guide* (CMU/SEI-2005-TN-015). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.
http://www.sei.cmu.edu/publications/documents/05.reports/05tn015.html

McGregor, John D. *Testing a Software Product Line* (CMU/SEI-2001-TR-022, ADA401736). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001.
http://www.sei.cmu.edu/publications/documents/01.reports/01tr022.html

Morell, Larry J. & Deimel, Lionel E. *Unit Analysis and Testing Curriculum Module (SEI-CM-9-2.0).* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1992.
http://www.sei.cmu.edu/publications/documents/cms/cm.009.html

SEI Education and Training: *Software Acquisition Survival Skills.* System Engineering Module. For more information about this course, visit http://www.sei.cmu.edu/products/courses/sass.html.

GUIDELINE 11: Software Risk Management

Contributor: Ray C. Williams

11

GUIDELINE

Software risk management is only successful when organizations are trained to actively identify risk, analyze and track it, plan how to mitigate it, communicate across all levels of the organization, and use risk information actively in making key decisions.

SCENARIO

A large IT acquisition program was building a centralized interactive database system to replace the current legacy system. Although the program was aware of risk management, they had not instituted it for the program. Some of the hindrances that the program encountered

- *"Don't Give Me Bad News"*: Upper level management refused to listen to concerns that meeting a deadline might be impossible. This culture led to a false optimism that things would "still work out" (despite evidence to the contrary) and an unwillingness to seek out information that would disprove the optimistic premise.
- *"Bow Wave" Effect*: The program developed the simple portions of the system first and showed good progress, but delayed key, complex development until the end. This created a "bow wave" of accumulated work to be done at the end, which increased the risk of missing deadlines.
- *Characterizing Total Risk*: The successful conversion of the data from the legacy system to the new system was essential to the success of the program. While aware of potential issues, the program neither evaluated the extent of them, nor prepared mitigation strategies.
- *Excessive Risk Avoidance*: The culture of the operational side of the organization made them very averse to proactively addressing risk because they did not want to disrupt support of the customer. However, since the program had no method for formally identifying, quantifying, or prioritizing risks, all they could do was accept risks as they arose, regardless of their magnitude.

Without risk management, the planners did not have to confront the growing risk to the program. Because they were unaware of the collective development risk they had taken on, they could not determine when the program would finish. The program ultimately faced the threat of cancellation because it repeatedly missed deadlines over an extended period.

ANALYSIS

In today's software-intensive systems, risks are often buried in the details, where decision-makers cannot see them. Furthermore, if a risk is not documented, it will be forgotten or neglected. Some keys to structuring an acquisition so that risk management will be successful in a program are

- Understand the difference between a *risk* (what *could* happen) and a *problem* (what *has* happened), noting that an existing problem can breed new risks from the decisions made to fix it.
- Create a program culture in which reporting a risk is not punished, but *rewarded*. If you "shoot" or even *ignore* one messenger, you will not get any information from them again.

- Realize that risk management is not a "check the box" activity, but something that helps program management to make good decisions throughout the program, and is thus something worth investing time and money in doing well.
- Know that effective risk management is a team activity that requires both active participation at all levels and good communication *across* those levels.
- Believe that *all* risk in a program can *always* be mitigated away; there is a point at which the total cumulative risk can become so great that it cannot be mitigated.
- Realize that managing risk requires staff training and expertise.
- Recognize that it is best to have program risks managed jointly between the PMO and contractor, but neither side should be forced to share information about *all* of their risks. For example, the PMO may choose to keep private the risk of a possible funding cut.

REQUIRED ACTIONS

- Start a risk management program on Day 1 of the program.
- Ensure that PMO staffs have appropriate risk management training, and if possible, experience to be able to identify both technical and non-technical risk items.
- Use multiple methods to identify risk sources, such as periodic risk reporting, brainstorming, voluntary risk reporting, risk report forms, or questionnaires or interviews based on the software risk taxonomy in the SRE *Method Description* (see "Implementation Resources" below).
- Include a risk management program in the RFP and contract that specifies how to report risks.
- Structure incentive/award fees to include risk management, and reward early risk identification.
- Promote decentralization of risk identification/analysis by using an organizationally defined process.
- Establish and maintain a schedule of joint risk reviews with all contractors throughout the program, including joint prioritization of the risks most important to the program.
- Consider conducting independent risk assessments on the program.
- Define a process and criteria for escalating risks.
- Establish a PMO risk management process, and put PMO risk mitigation plans in place.
- Review the acquisition strategy for programmatic risks.
- Ensure risk management tasks/strategy align with development and acquisition strategies.
- Communicate regularly with the contractor about program risks and status.

DANGER SIGNS

- Risks are recorded in different forms, are unclear, unactionable, or too high level
- The quantity of risks in the repository remains stagnant
- Program personnel mention issues that concern them, but they are not captured and added to the risk repository
- Program personnel do not know how to enter a risk into the repository, or are afraid to
- There are no regular risk management meetings
- In program reviews, the risk portion is often omitted due to "lack of time"
- The status of documented risk items never changes
- There is no owner assigned to assess and mitigate a given risk
- The escalation path for risks is unclear
- Identified risks do not have associated mitigation strategies (even if the strategy is to simply "watch" the risk over time)

IMPLEMENTATION RESOURCES

Defense Acquisition University (DAU). *The Risk Management Guide for DoD Acquisition, Fifth Edition.* Fort Belvoir, VA: Defense Acquisition University Press, June 2002.
http://www.dau.mil/pubs/gdbks/risk_management.asp

Dorofee, Audrey; Walker, Julie, Alberts, Christopher; Higuera, Ronald; Murphy, Richard; & Williams, Ray. *Continuous Risk Management Guidebook.* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1996.
http://www.sei.cmu.edu/publications/books/other-books/crm.guidebk.html

SEI Education and Training: *Continuous Risk Management.* For more information about this course, visit http://www.sei.cmu.edu/products/courses/crm.course.html.

SEI Education and Training: *Software Acquisition Survival Skills.* Risk Management Module. For more information about this course, visit http://www.sei.cmu.edu/products/courses/sass.html.

Williams, Ray C.; Pandelios, George J.; & Behrens, Sandra G. *SRE Method Description (Version 2.0) and SRE Team Members Notebook (Version 2.0)* (CMU/SEI-99-TR-029, ADA001008). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1999.
http://www.sei.cmu.edu/publications/documents/99.reports/99tr029/99tr029abstract.html

GUIDELINE 12: Software Metrics

Contributor: Software Engineering Measurement and Analysis Initiative

12

GUIDELINE

Planning effective use of software acquisition metrics involves identifying appropriate metrics for the program, requiring them in the contract, monitoring them frequently, and acting on them early and decisively.

SCENARIO

Several types of program failure can be related to measurement or specifically, the lack thereof. One of the most common errors is that a program assumes that a single metric (such as earned value management, or EVM) is sufficient for monitoring multiple aspects of the program.

For example, an entire program can go "off track" because of a delay caused by a single team. One program experienced this problem when government furnished equipment (GFE) was promised to the vendor, but was delivered nearly 7 months late. Since the technology was new to the development team, the adoption and integration of this equipment was on the project's critical path. For months, EVM showed only a minor slip in the schedule, but when the integration date arrived this single point of failure forced the entire program to re-baseline its plans.

Schedule pressures can also cause a program to fail. During "crunch times," a development team began to take shortcuts in their processes. The PMO was not aware of these process changes because it did not have access to quality assurance measurements. As a result, extreme delays and cost overruns plagued the testing phase because the "shortcuts" propagated defects throughout the project.

Each of these examples demonstrates how general risks to a program can be monitored by using specific measurement data, revealing problems that EVM alone cannot. More detailed data gives the PMO a stronger point of leverage when it comes time to take action.

ANALYSIS

Effective use of metrics in an acquisition program requires advance planning. Choose metrics that reveal the program's status at a high level, but also use metrics that allow you to closely monitor the areas of highest risk to the program. Incorporate measurement requirements into the RFP and contract, so that the vendor must provide the data you need, in the form you want. Train the PMO staff to interpret the metrics so that they can use the data being collected.

After the program gets underway, examine the following detailed metrics data early:

- Ensure that the metrics you have identified for the program continue to be collected in a consistent, reliable way—misleading data can be more dangerous than no data at all. Analyze *multiple* independent indicators to be sure that the data are consistent.
- Plan to review cost, schedule, size, and effort data early and frequently (weekly). Doing so can reveal problems early, when they can be handled more effectively and less expensively.

- Use historical data to determine how much deviation or variance in a metric is enough to warrant action. This determination depends on past experience and factors such as project size, project type, project phase, and so on.
- When you observe deviations in the metrics, get more detailed information and follow the data to the *root cause*. This not only allows you to correct it properly (as opposed to treating only the symptom), but to put preventative measures in place for the future.

Programs do not get behind overnight—they slip day by day. The key to avoiding major schedule slips is early warning. Finding the root cause *behind* the warning gives insight for informed decision-making and objective, fact-based planning, managing, and communicating. After you have identified the true cause behind an issue, define a realistic course of action and take it.

REQUIRED ACTIONS

- Fine-tune measurement expectations (and RFPs) by better understanding contractor processes, taking advantage of review opportunities, and identifying risk indicators up front.
- Hire and/or develop PMO measurement expertise and analysis skills to avoid pitfalls like analyses that miss the "big picture," charts that are colorful, but meaningless, predictions drawn from data on unstable development processes, and contractors who "spin" the data being presented.
- Monitor acquisition processes such as requirements development, contract management, and the oversight process itself.
- If you are in an early program phase, determine the metrics required in the contract based on program risk areas and metrics you will collect to monitor PMO performance, and require training, warfighter, and logistics commands to participate.
- If you have not yet released your RFP, ensure the required metrics are consistent with the acquisition strategy, link your incentive fee or award fee to appropriate metrics, and ensure that the RFP specifies timeliness of metric reporting.
- If you are still negotiating the contract, include flow-down to subcontractors, ensuring that they contribute to the requested metrics. Also understand how the contractor uses metrics so you can use their current practice, which can also lower cost.
- If you are executing the program:
  - Evaluate your metrics to ensure that you are getting the data you need to oversee the program
  - Ensure that all the metrics are consistent with observable progress
  - Ensure that you are using metrics to monitor the performance of the PMO
  - Do not allow program's change management process to cause the EVMS to "lose" program history
  - Be an active, questioning participant in all contractor reviews
  - When being presented with data, *always* ask, "Has the measurement system changed?"

DANGER SIGNS

- A *single* metric is being used to monitor program status
- No PMO measurement plan exists either for evaluating contractor performance, or for internal PMO measures
- The RFP or contract do not specify delivery of relevant measurement data
- No PMO staff is trained in measurement definition and analysis
- Excessive amounts of program metrics are supplied that do more to confuse than to inform
- The PMO cannot analyze and respond to the measurements/metrics provided by the contractor
- The PMO is not connected with the other external stakeholders through measurement data
- Measurement data is:
  - inconsistently collected, late, significantly imprecise, inaccurate, changing/unstable, or not presented in a useful graphical format
  - not linked to an understood process (either *ad hoc* or defined)
  - disconnected from clearly defined goals
  - lacking in any baseline for comparison
  - represented in a different format each time it is presented

IMPLEMENTATION RESOURCES

Brownsword, Lisa & Smith, Jim. *Using Earned Value Management (EVM) in Spiral Development* (CMU/SEI-2005-TN-016). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005. http://www.sei.cmu.edu/publications/documents/05.reports/05tn016.html

SEI Education and Training: *Software Acquisition Survival Skills.* Managing with Metrics Module. For more information about this course, visit http://www.sei.cmu.edu/products/courses/sass.html.

GUIDELINE 13: Software-Based Award Fees

Contributor: Julie B. Cohen

GUIDELINE

Carefully consider the desired behavior that may be encouraged by an award fee, the possible consequences of encouraging that behavior and not others, and the exact way in which the award fee is worded.

SCENARIO

A large IT system acquisition was being developed under a Firm Fixed-Price (FFP) contract[15] and the government was under substantial pressure to deploy the system to its field sites as quickly as possible. Because of this, the contract included a single award fee that stated, "The Government wishes to incentivize the contractor to come in early with a 1% award fee." As the delivery date came closer, there was a headlong rush to say that everything was ready to go and to deliver the system by the end of the performance period on November 30. The contractor delivered the system on November 13, only two weeks early, and received an award fee payment of $600,000.

Within a few weeks of having been put on-line, the system failed spectacularly, causing the pilot site to nearly shut down operations and creating an enormous additional workload for the employees at the site who were responsible for operating the system. The users felt that they had been "...stuck with all of the system problems, just so that the contractor could get their bonus."

ANALYSIS

Because they are subjectively evaluated, software-based award fees can be used to emphasize less measurable deliverables such as the quality of the architecture and the software design documentation. Also, these objectives can change from award fee period to award fee period and are commonly tied to events (for example, "Successful completion of <X>"), rather than to schedule (for example, "Completion by <date>"). They have been shown to be successful in enhancing software development and documentation quality.

As illustrated by the scenario, award fees can result in unintended consequences. Key issues to be aware of include:

- Incentivizing one attribute can "de-incentivize" others. Put yourself in the contractor's position and think about how the contractor might interpret and respond to the incentive. Word incentives very carefully.
- Sometimes the contractor may put so much effort into preparing for the award fee board that productivity is sacrificed. Also, problems that should be dealt with as a team may be hidden from the government so that the contractor looks good on the performance reports.

15 Note that this is not a recommended approach for this type of development effort—it evolved from a prior relationship with the contractor in a smaller consulting role.

- Incentivizing defect detection during inspection or test can have the unintended effect of encouraging an unscrupulous contractor to insert additional "defects" during coding that will be "found" later during inspection/test.
- Contractor program managers may be evaluated by their company based on award fee performance—and removed from the position if performance drops below a predefined percentage, which in practice may be as high as 80%. Be aware of this when assigning final award fee percentages.
- Most commercial organizations want to earn at least a 20% profit. When negotiating a contract with, for example, an 8% fixed fee and a 3% award fee, the contractor may not be highly motivated—especially if the contractor does not normally work under government contract.
- Ensure that all of the PMO staff are aligned with the objectives of an award fee. Disagreements as to the value of an award fee's purpose can create problems when it comes time to evaluate the contractor's performance.
- Planning and evaluating award fees requires PMO time and effort—ensure that the appropriate staff and schedule are available to effectively execute an award fee plan.

REQUIRED ACTIONS

- Ensure that the award fee base fee percentage does not exceed the FAR prescription of 3% (even a FAR deviation cannot change this).
- For Defense agencies, ensure that the base fee does not exceed 3% of the estimated cost of the contract exclusive of the fee (DFARS 216.470).
- Award fee evaluation requires effort. Ensure that adequate PMO staff time is reserved to do it properly.
- Consider award *term* where the contractor earns additional periods of performance (contract extensions) as an alternative to award fee. This approach can be especially effective for support and sustainment contracts.
- Evaluate software development progress in conjunction with overall system development progress to avoid achieving goals in one area at the expense of the other.
- Consider changing the award fee plan each period if necessary to emphasize what is important at that particular point in the program.
- One program had several different contracts and the contractors needed to work together to integrate a large system. The PMO decide to use three different award fees. One was a standard award fee based on a six-month plan. The second was a mission success award fee based on successful completion of large program milestones (CDR, DT&E completion, operational activation, etc.). The third part was a shared award fee. This portion was earned only if the entire team of several prime contractors worked together to meet specific award fee criteria. If your program manages several inter-related contracts you may want to consider this.

DANGER SIGNS

- The contractor overemphasizes meeting an award fee for quality, which could manifest itself as shortfalls in non-award areas such as performance and schedule. Monitor the status of these other areas very carefully
- Identification of risks slows noticeably during the period prior to an award fee evaluation

IMPLEMENTATION RESOURCES

Averill, E.; Byrnes, P.; Dedolph, M.; Maphis, J.; Mead, W.; & Puranik, R. *Software Capability Evaluation (SCE) Version 1.5 Method Description* (CMU/SEI-93-TR-017, ADA267895). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993. http://www.sei.cmu.edu/publications/documents/93.reports/93.tr.017.html

SEI Education and Training: *Software Acquisition Survival Skills*: Pre-Award Activities Module. For more information about this course, visit http://www.sei.cmu.edu/products/courses/sass.html.

Acronyms

ATAM: Architectural Tradeoff Analysis Method

ATO: Authority to Operate

C4ISP: Command, Control Communications, Computers, and Intelligence Support Plan

CCEVS: Common Criteria Evaluation and Validation Scheme

CD: Compact Disc

CDR: Critical Design Review

CLS: Contractor Logistics Support

CMMI: Capability Maturity Model Integration

CMU: Carnegie Mellon University

CORBA: Common Object Request Broker Architecture

COTS: Commercial Off-The-Shelf

CPI: Critical Program Information

CSCI: Computer Software Configuration Item

CSIA: Cyber Security Industry Alliance

CSR: Critical System Resources

DAA: Designated Approving Authority

DAU: Defense Acquisition University

DFAR: Defense Federal Acquisition Regulation

DIACAP: Defense Information Assurance Certification and Accreditation Process

DISA: Defense Information Systems Agency

DITSCAP: Defense Information Technology Security Certification and Accreditation Process

DoD: Department of Defense

DoDI: Department of Defense Instruction

DT&E: Developmental Test and Evaluation

EVM: Earned Value Management

FAR: Federal Acquisition Regulation

FFP: Firm Fixed Price

FISMA: Federal Information Security Management Act

GFE: Government Furnished Equipment

GNO: Global Network Operations

GUI: Graphical User Interface

IA: Information Assurance

IASE: Information Assurance Support Environment

IATAC: Information Assurance Technology Analysis Center

IAVM: Information Assurance Vulnerability Management

IT: Information Technology

IV&V: Independent Verification and Validation

JTF: Joint Task Force

MAC: Mission Assurance Category

MIS: Management Information System

MOSA: Modular Open Systems Approach

NIAP: National Information Assurance Partnership

NIST: National Institute of Standards and Technology

NSA: National Security Agency

OCTAVE: Operationally Critical Threat, Asset, and Vulnerability Evaluation

OT&E: Operational Test and Evaluation

PM: Program Manager

PMO: Program Management Office

QAW: Quality Attribute Workshop

RFP: Request for Proposal

SEI: Software Engineering Institute

SEMP: Systems Engineering Master Plan

SORAP: Source of Repair Assignment Process

SPCA: Security Policy Compliance Assessment

SRE: Software Risk Evaluation

SSAA: System Security Authorization Agreement

TEMP: Test and Evaluation Master Plan

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE December 2005	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Software Acquisition Planning Guidelines		5. FUNDING NUMBERS FA8721-05-C-0003
6. AUTHOR(S) Julie B. Cohen, Anthony J. Lattanze, Linda Levine, PhD, William E. Novak, Patrick R. H. Place, Ray C. Williams, Carol Woody, PhD		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213		8. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-2005-HB-006
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS		12B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

Guidance about acquisition planning and strategy is scattered by topic throughout many different books, reports, presentations, and Web sites. This handbook presents guidance for acquisition planning and strategy topics in a condensed form, and references the primary resources available for each topic.

The topics for the guidelines in this handbook represent selected areas in which the SEI has conducted significant research. The guidelines are organized by the acquisition strategy considerations categories from the Defense Acquisition University's Defense Acquisition Guidebook that describe the principal areas of consideration for planning an acquisition strategy. Guidance is offered on 13 topics, including open systems approach, commercial off-the-shelf (COTS)-based systems, software architecture, software sustainment, requirements development, requirements management, operational information assurance, information assurance for COTS sustainment, information asset protection, software testing, software risk management, software metrics, and software-based award fees.

This handbook is intended to provide program managers and project management office staffs with recommendations and resources for addressing different aspects of their acquisition strategy. It illustrates acquisition challenges with program scenarios, suggests actions that should be taken, and identifies danger signs to look for when evaluating progress. A basic knowledge of both software development and acquisition practices is assumed.

14. SUBJECT TERMS acquisition, COTS, interoperability, metric, software-intensive system, security, sustainment, risk management, software requirements, process, testing, systems engineering			15. NUMBER OF PAGES 62
16. PRICE CODE			
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL